Application of Knowledge Acquisition Techniques to Requirements Capture: A Case Study

Jorge Enrique Pérez-Martínez and Almudena Sierra-Alonso

Universidad Politécnica de Madrid
Departamento de Informática Aplicada
Campus Sur de la U.P.M., 28031 Spain
email: jeperez@eui.upm.es

Universidad Rey Juan Carlos
E.S.C.E.T.
C/Tulipán s/n, 28933 Móstoles, Spain
email: asierra@escet.urjc.es

Abstract

In this paper we want to point out, by means of a case study, the importance of incorporating some knowledge engineering techniques to the processes of software engineering. Precisely, we are referring to the knowledge eduction techniques.

We know the difficulty of requirements acquisition and its importance to minimise the risks of a software project, both in the development phase and in the maintenance phase. To capture the functional requirements use cases are generally used. However, as we will show in this paper, this technique is insufficient when the problem domain knowledge is only in the "experts' mind". In this situation, the combination of the use case with eduction techniques, in every development phase, will let us to discover the correct requirements.

1. Introduction

Kotonya and Sommerville [4] establish that "the requirements are descriptions of how the system should behave, application domain information, constraints on the system's operation, or specifications of a system property or attribute. Sometimes they are constraints on the development process of the system" (p. 6). Jacobson, Booch and Rumbaugh [3] indicate that "a requirement is a condition or capability to which a system must conform" (p. 448).

Since the requirements capture is a critical task for the development and maintenance of an application, large efforts are being realised to delimit an area that has been called "requirements engineering" (RE). Thus, for example, the Software Engineering Co-ordinating Committee [9], in the SWEBOK (Software Engineering Body of Knowledge) establishes six areas in the process of RE: requirements engineering process, requirements elicitation, requirements analysis, software requirements specification, requirements validation and requirements management (pp. 2-10 to 2-19).

Here we are interested in the problems associated with the requirement elicitation. The mentioned authors agree that the requirement acquisition is a difficult activity. The reasons are:

- The customers are an imperfect source of information.
- The customers express requirements in their own terms and with implicit knowledge of their own work.
- When the requirements have been closed, making changes is very expensive.
- It is difficult to establish if the requirements are complete and consistent.
- The cultural diversity of the people who participate in the systems software construction.

It seems a common practice to capture the functional requirements using use case. What is not indicated by any of the authors is how to understand the problem domain. The Unified Process [3] indicates that a domain model or

a business model has to be built. In both cases, if the information to build the model is only known by the experts, interacting with them is suggested to elicitate that information. Thus, the Software Engineering Co-ordinating Committee [9] proposes the following techniques for the requirements capture: interviews, scenarios, prototypes, meetings (brainstorms, conflicts resolution, etc.) and observation of the users' tasks (pp. 2-12 to 2-13). Maiden and Rugg [5] add other techniques: protocol analysis, card sorting, laddering, repertory grids, RAD workshops and ethnographic methods. More detailed information about these techniques and how they are applied can be found in [6,8]. This set of techniques can remind us of some othes used in knowledge engineering (KE). In fact, in [1] we can find a list of KE activities that are applied to SE (and vice-versa). One of those activities is knowledge acquisition. In this sense, we think the knowledge eduction techniques used in KE are ideal for the requirements elicitation task, as the case study will illustrate. Maiden and Rugg [5] already indicate that RE and KE share many features and they propose some heuristics to use KE techniques for the requirements capture. But that work deals with the requirements acquisition as an isolated task. That is, they do not embed the problem in a development process. In this work we use some of the mentioned knowledge eduction techniques, but we do not follow any particular KE development process.

The aim of our work is to determine: a) when it is necessary to use KA techniques more sophisticated than those we usually use in SE; and b) which technique is better in each moment. To achieve this objective we plan to analyse several case studies to detect where are there lacks or difficulties in the requirements capture. In this paper we are presenting the results of the first of those projects. To develop this project we use the Unified Process which is a well-defined, successful and contrasted process. We think that it can be very useful to add to this Process some guidelines to help the software engineer in the requirements capture phase.

In Section 2 we present the case study in which we have combined SE techniques and KE techniques. In Section 3 we describe the first use case model obtained during the inception phase and the problems to realisation some of them. In Section 4 we illustrate the knowledge eduction process, using KE techniques to obtain the rules that the expert used to solve the problem. Section 5 illustrates in which Unified Process phases and activities the KE eduction techniques have been used. Finally in Section 6 we present the conclusions of this work and future developments.

2. The project: "Similarity Detection in Programs Written in Smalltalk"

The Department of Informática Aplicada of the Polytechnic University of Madrid teaches the course "Object-Oriented Software Components". To pass this course, the students have to do some exercises using the Smalltalk programming language. Software that detects if those exercises are copied is required. The client provided us the following specifications [7]:

"An application is required to determine which programs, that solve the same problem, possess a similarity degree over a certain threshold. The application has to generate a list with those exercises that apparently have been copied. This list will be used by the responsible of the grading as a possible sign of copying. Then, the responsible will have to check manually the programs indicated as copied. Thus, the application will be a filter to help to the grading task.

The application has to be adaptable by means of changing some data such as the similarity degree from which an exercise is considered copied or varying the importance of some parameters used to detect a copy. Sometimes the application does not have to take into account some features because those characteristics are imposed by the exercise, so we have to be able to indicate these features to the application. The application must compare all students' exercises, including exercises from previous courses. The application will be executed in a Pentium 133Mhz with 32Mb of RAM and the execution time must be less than 12 hours (a night). Both batch and interactive execution are required. The resulting application should be "aggressive". That is, it is better for the application to indicate as copied exercises that are actually not copied, than passing some possibly copied exercises. The application has to behave at least as well as the expert and the explanations given by the application to justify why two exercises are declared as copied have to be short and meaningful. The system must be working before November, 1999."

The project began in March 1999. The Unified Process [3] was used as development process and UML [2] as notational language. We are not going to detail every activity executed throughout every phase; we will only describe those where the knowledge eduction techniques introduction have been vital for the application development. The detailed information can be found in [7].

The specification document elaborated by the client contains functional and not functional requirements. In this paper, we will focus on the functional requirement capture and, specifically, in the exercise comparison process to detect whether they are copied.

3. The initial capture of functional requirements using SE techniques

To capture functional requirements we used the use cases. With the specifications and informal interviews with the client we developed a first use case model (generated during the requirements acquisition activity in the inception phase) that is illustrated in figure 1. To realise "Configure Application" and "Execute Application" use cases we did not have enough knowledge. For "Configure Application" we did not know what elements were going to de considered and for "Execute Application" we did not know what class attributes should be taken into account, how they depended on each other and their relative importance. The "Present Results" use case was not completely understood because we did not know what information the client required neither the detail of that information. We thought that this one would not be a problem because when we knew the process to determine whether an exercise is copied, to structure the analysed information and to present it according to the client's needs would be immediate. For the "Submit Exercise" use case we had a meeting with the client. He indicated the required functionality and we reached an agreement on the user interface. In fact, a prototype (without functionality) was constructed for the client to validate the presentation and navigation aspects.

In this paper, we will focus on the realisation of the "Execute Application" use case. Figure 2 illustrates the analysis classes for a realisation of this use case. In this figure, the class DSPS (Similarities Detector in Programs written in Smalltalk) is the copy detector, and the rest of this paper will focus on it.

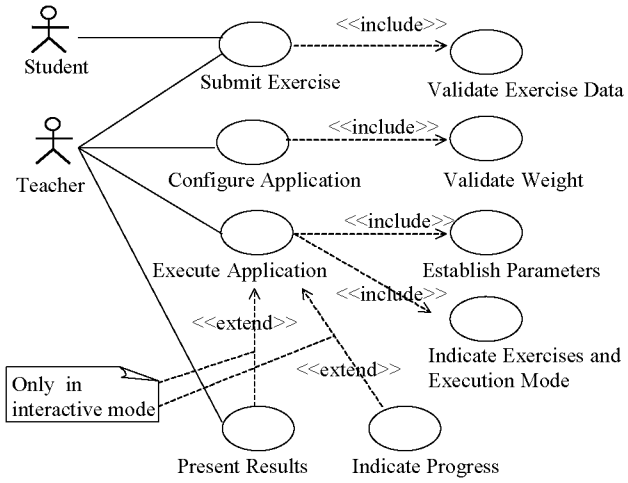

Figure 1. Use case model in the inception phase.

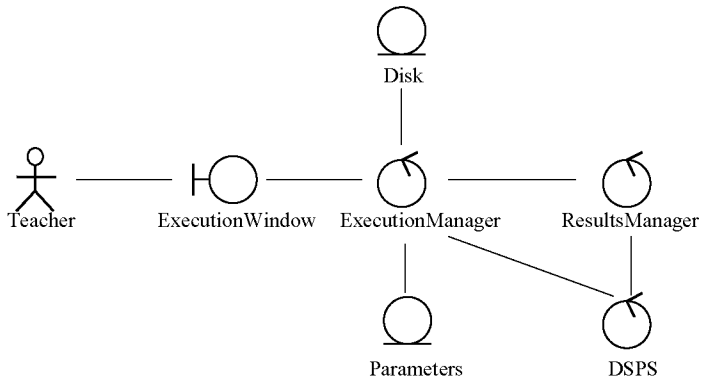

Figure 2. A class diagram for a realisation of the "Execute Application" use case.

4. The knowledge acquisition process for the copies detection

The most interesting facet in the "Execute Application" use case is to determine which exercises are copied; that is, the interest is the analysis class DSPS. It was necessary to find out which criteria the expert used to conclude that two exercises were copies and which mental process he followed when he applied those criteria. At the beginning, we noticed that we did not have knowledge about the Smalltalk language nor the type of exercises that the client asked to his students. We ignored the criteria used by the expert to take his decisions, the process to apply those criteria, every criterion weight and how the weights were combined.

The first two problems were easy to solve. We got documented about the Smalltalk language and we asked the expert for exercises. With the acquired knowledge we could understand partially the problem domain. But we did not find documentation to solve the third issue. We had to acquire the knowledge from the expert. At this point we decided to use the knowledge eduction techniques of the KE.

We noticed we did not have automatic tools to carry out the eduction process. This one was a restriction for the use of techniques such as the repertory grid analysis. On the other hand, as we had a unique expert, we dismissed the group eduction techniques (such as brainstorm or Delphi method). We thought that to apply inducement eduction techniques, over all the family TDIDT (Top Down Induction of Decision Trees) could be interesting. We had to verify if the expert had enough cases, if they were classified and if he could distinguish the primary attributes that would be the process base. This technique, like the repertory grid analysis, permits to do classifications. From a global point of view, the similarity detection between programs is basically a classification of them in two categories: copied exercises and not copied exercises. Although the expert had many test cases, they were not classified and a set of criteria to classify them did not exist.

On the other hand, we dismissed the use of the "observation of the task performance" and "critical incidents" eduction techniques because the expert was not executing the task in the moment the acquisition had to be carried out (the expert executes the task in February, June and September). He was reticent to support the application development on his long-term memory. Thus, we decided that we were going to use the following eduction techniques: structured and unstructured interviews (recommended by SE too), questionnaires and protocol analysis.

4.1. What criteria does the expert use to determine that two exercises are copied?

We had a first meeting with the expert using the open interview technique in order to obtain general information about the problem and its context. The results of analysing the meeting were the following:

1. We began to guess what the expert meant when he was speaking of the structural and behaviour analysis of every class.
2. We began to see the class features that the expert observed to carry out the analysis.
3. We obtained the user interface.
4. We concluded that the similarity degree among exercises was a combination of the class similarities. What was difficult to establish is the similarity degree between classes.

4.2. How are two classes compared?

To determine the similarity degree between classes a second eduction meeting with the expert was held. In this case, we used the structured interview technique. In this interview we concluded that the analysis of similarities between two classes is approached from three different and orthogonal points of view. Every point of view is a subproblem to solve. These points of view are the following:

- Lexicographical analysis.
- Structural analysis, that refers to the number, name, instantiation, etc. of the data structures used.
- Behaviour analysis referred to the methods that contains each class and their number, type, arguments, comments and body. There is another deeper semantics analysis referred to the expressions, control sentences, instantiations, etc. of the method body.

With this information, we could complete the use case model described in figure 1 during the requirements capture activity in the elaboration phase. Figure 3 shows the part of this model related to the "Execute Application" use case. As we can observe, we already identify other use cases related to the parameterisation of the application execution (indicated by the client) as well as others related to the explanations that the application would have to provide. The use cases in which we are interested are shaded in grey and they are obtained from the analysis of the two eduction meetings accomplished.

We already had determined that the expert carried out the class analysis at different detail levels. However we

did not know yet which factors we have to consider for this analysis. Then, we planned two more eduction meetings. In the first one, we used the questionnaire technique while in the second one we employed the protocol analysis facing the expert with a real case of two classes that he was considering copied. We will not detail all the obtained products from these meetings analysis. We will only describe the aspects related to the protocol codification.

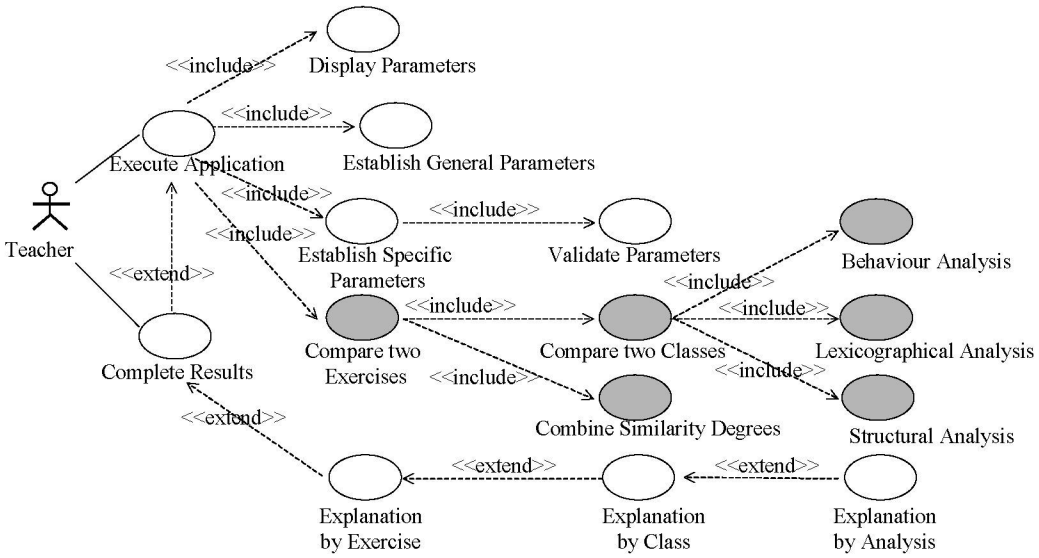

Figure 3. Subordinated use cases to the "Execute Application" use case.

4.2.1. Protocol Codification

In this phase, the most relevant aspects are related to the identification of search and operators. After analysing the protocol, we can see that the expert executes the following steps when he compares two classes:

1. Class names: if they are the same, and they are not environment parameters, the certainty that the classes have been copied increases a bit (hypothesis).
2. Class variables, *pool-dictionaries* and *instance variables indexed*: if there is any of them in a class but not in the other, the hypothesis certainty degree decreases a lot.
3. Number and names of *instanceVariableNames*: If they are the same in number and name, the hypothesis certainty degree increases a lot. If only the number is the same, the expert establishes the hypothesis that the identifiers have been permuted inside the methods. This hypothesis must be confirmed by analysing the methods.
4. Class methods: if they are the same, the hypothesis certainty degree is very high.
5. Instance methods: if the number is the same the hypothesis certainty degree increases a lot.
6. Every instance method of a class is compared with every instance method of the other class. This comparison is realised in four steps:
   - 6.1. Headers comparison: names and number of arguments. If names are not the same, the expert carries out a substitution process: if the method is binary, he looks for a keyword method with only one argument in the other class. If the argument names are different, the expert establishes the hypothesis of permutation described in 3, but with the identifiers of the arguments.
   - 6.2. Comments: a lexicographical analysis is performed with both comments, taking into account the substitution hypothesis described above.
   - 6.3. Temporary variables: he compares the number, the names and the apparition order. If the number is the same, the hypothesis certainty degree increases. If the names are not the same,

the expert permutes the identifiers as with the method arguments.

6.4. Body: if necessary, the expert realises the permutation of the identifiers (instance variables, arguments and temporary variables). Then, he analyses the syntactic tree and some control structures.

The operators identified are:

- **Comparison** operator. It compares different objects and its result increases or decreases the hypothesis certainty degree. To carry out this comparison, the expert uses a **search** operator to detect non-commonly used methods, data and control structures.
- **Substitution** operator. This operator substitutes identifiers of instance variables, method arguments and temporary variables.
- **Equivalence** and **inverse equivalence** operators. They are used to transform a method that implements a binary message in another one which implements a keyword message with a unique argument and vice-versa.
- **Quantification** operator. This operator quantifies objects, instance methods, *instanceVariableNames*, temporary variables and method arguments.
- **Elimination** operator. This operator eliminates those features that do not contribute to the analysis. For example, environment parameters, spaces, carriage returns, *accessor* and *mutator* methods, etc

The search strategy and operators allowed specifying the factors that took part in the analysis of two classes. Figure 4 illustrates a class diagram for a realisation of the "Behaviour Analysis" use case, that includes the results obtained in this protocol.

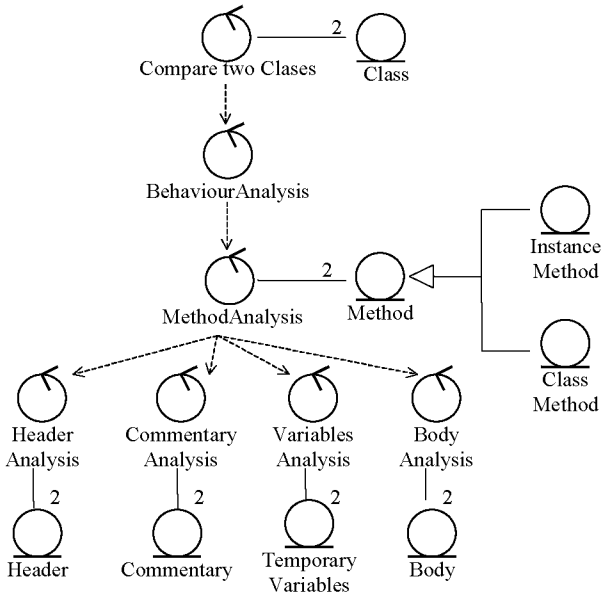

Figure 4. A class diagram for a realisation of the "Behaviour Analysis" use case.

5. Knowledge Eduction Techniques in the Unified Process Framework

To summary, figure 5 illustrates in which phases and activities the SE techniques and the knowledge eduction techniques proposed by the KE have been used. The numbers into a circle refer to products obtained by the SE techniques application. The numbers into a star identify products obtained by the KE techniques application. Following we describe the product identified by each number:

1. Use case Model (figure 1).
2. A class diagram for a realisation of the "Execute Application" use case (figure 2).
3. Subordinated use cases to the "Execute Application" use case (figure 3). They were discovered from the two first acquisition meetings analysis, which were realised using the open and structured interview techniques, respectively.

4. Terms glossary, concept/attribute/value table, intermediate representations and relationships between concepts. These products were direct consequence of the third meeting analysis using the questionnaire technique.

5. A class diagram for a realisation of the "Behaviour analysis" use case (figure 4). It was obtained from the fourth eduction meeting analysis carried out using the protocol analysis technique.

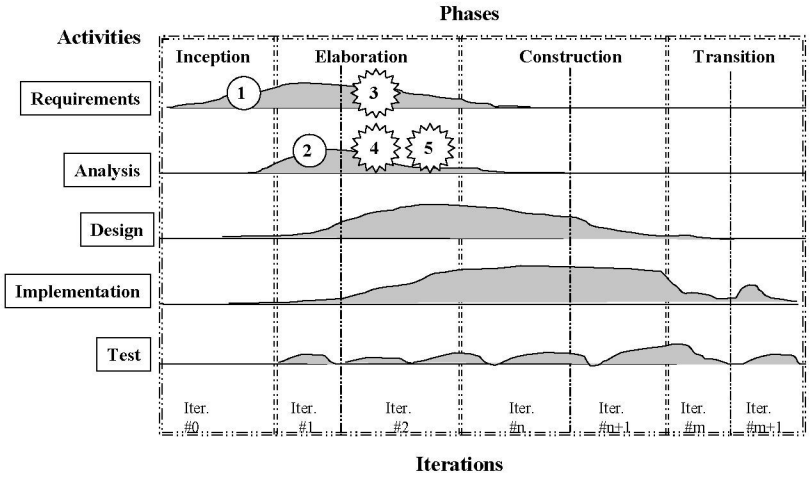

Figure 5. Use of SE and KE techniques during the project development.

In spite of different applications having different necessities with regards to the moment of using knowledge engineering techniques, we think it is reasonable to establish the following guides to use these techniques inside the Unified Process (while waiting for the conclusions of study set out in section 6):

1. The use of these techniques does not seem necessary during the inception phase (in this phase feasibility of the project is established, risk analysis is carried out and system limits take shape).

2. The objective of elaboration phase is to establish the system architecture. Where eduction techniques can be used more widely is in this phase. They will help us to shape the subsystem architecture. Specifications usually give little information about them. Concretely, for bad defined subsystems, the use of the eduction techniques during requirements capture and analysis activities can provide refinements of the use cases and class diagrams for realisations of these use cases.

3. The construction phase leads to initial operational capability; this one involves low level design aspects. The eduction techniques can help to clarify some architecture components or the relationships between them that have not been elaborated yet.

4. In the transition phase, applying these techniques will not be necessary, unless functionality errors or misinterpretations of the specifications appear during the test stage with the beta users (and before deployment).

6. Conclusions and Future Works

In this paper we have shown that the knowledge acquisition techniques of the KE are not only suitable for some SE phases, but sometimes they are indispensable. Surely, the requirements capture is the most critical activity for any software application; the KE eduction techniques permit us to reduce the risks of this activity.

In this work we have identified high-level architecture aspects directly derived from the specifications document and from the informal interviews with the client. However, when we have to define the subsystems and their components, and we do not have knowledge of the problem domain, it is indispensable to turn to the use of well-known knowledge eduction techniques that we can not find in the modern SE. In the described case study, the specification for the subsystem that realises the "Execute Application" use case was obtained using knowledge eduction techniques. The results of the three first sessions with the expert, such as the term glossary or the concept/property/value table, let us obtain a preliminary approach to the problem space objects and their relationships. The fourth session, performed using the protocol analysis technique, gave many results. From this

session we obtained, for example: the steps that the expert followed to search the similarities, the entities that he compared, the operators which he used to do that comparison and the criteria that he applied to combine the comparison results in order to obtain a final conclusion.

We think it would be interesting to get in touch with the co-ordinator committee of the SWEBOK in order to propose the discussion of this issue during the second subphase of the Iron Man version. The concrete proposal is the convenience of introducing the KE eduction techniques in the requirements capture area. We are also studying how to incorporate formal and methodologically such techniques to the Unified Process. For example: to establish in which development moment of the Unified Process to introduce the KE eduction techniques; which techniques are appropriate in which phases; and to redefine the set of activities to accomplish in every phase, the set of artifacts to generate, the involved workers and their responsibilities, the risks calculation and the effects in the project planning (costs and time).

References

[1] Acuña S., López M., Juristo N., Moreno A. (1999). A process model applicable to software engineering and knowledge engineering. *International Journal of Software Engineering and Knowledge Engineering*, 9, 663-687.

[2] Booch G., Rumbaugh J., Jacobson I. (1999). *The unified modelling language user guide*. Reading, MA: Addison-Wesley.

[3] Jacobson, I., Booch, G., Rumbaugh, J. (1999). *The unified software development process*. Reading, MA: Addison-Wesley.

[4] Kotonya, G., Sommerville, I. (1998). *Requirements engineering: processes and techniques*. West Sussex, Inglaterra: John Wiley & Sons.

[5] Maiden, N., Rugg, G. (1996, May). ACRE: Selecting methods for requirements acquisition. *Software Engineering Journal*, 11(3), 183-192.

[6] Olson, J.R., Rueter, H., (1988) Extracting Expertise fromExperts: Methods for Knowledge Acquisition, *Heuristics, The Journal of Knowledge Engineering*, 1(2), 21-41.

[7] Pérez-Martínez, J.E. (1999). *Detector de similitudes en programas escritos en Smalltalk [Similarity Detection in Programs Written in Smalltalk]*. Proyecto Fin de Carrera, Universidad Carlos III de Madrid, España.

[8] Scott, A.C., Clayton J.E., Gibson, E.L. (1991) *A Practical Guide to Knowledge Acquisition*. Reading, MA: Addison-Wesley. Reading, USA.

[9] Software Engineering Coordinating Committee (2000, April). *SWEBOK, a stone man version (version 0.7)*. Montreal, Canada: IEEE.